

---

# Learning Efficient Representations for Reinforcement Learning

---

Yanping Huang  
University Of Washington  
huangyp@cs.washington.edu

## Abstract

Markov decision processes (MDPs) are a well studied framework for solving sequential decision making problems under uncertainty. Exact methods for solving MDPs based on dynamic programming such as policy iteration and value iteration are effective on small problems. In problems with a large discrete state space or with continuous state spaces, a compact representation is essential for providing an efficient approximation solutions to MDPs. Commonly used approximation algorithms involving constructing basis functions for projecting the value function onto a low dimensional subspace, and building a factored or hierarchical graphical model to decompose the transition and reward functions. However, hand-coding a good compact representation for a given reinforcement learning (RL) task can be quite difficult and time consuming. Recent approaches have attempted to automatically discover efficient representations for RL.

In this thesis proposal, we discuss the problems of automatically constructing structured kernel for kernel based RL, a popular approach to learning non-parametric approximations for value function. We explore a space of kernel structures which are built compositionally from base kernels using a context-free grammar. We examine a greedy algorithm for searching over the structure space. To demonstrate how the learned structure can represent and approximate the original RL problem in terms of compactness and efficiency, we plan to evaluate our method on a synthetic problem and compare it to other RL baselines.

## 1 Introduction

This report considers sequential decision making problems where decisions can have both immediate and long-term effects. Each decision results in some immediate reward or benefit, but also affects the environment in which further decisions are to be made and thus affects the expected reward incurred in the future. The objective of the decision maker is to choose decision making policies optimally, that is, to maximize some long-term cumulative measurement of rewards. Such objective is challenging mainly because of the tradeoff between upfront and future rewards. Markov decision processes [32, 24] (MDPs) provides a mathematical formalization for this tradeoff.

### 1.1 Markov Decision Process

A MDP is mathematically defined in terms of a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ , where

- $\mathcal{S}$  is the finite set of all possible states that describes the context of the environment, also called the *state space*;
- $\mathcal{A}$  is the finite set of all actions the decision making agent can take;
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is a transition function, a mapping specifying the probability  $P_{s,s'}^a$  of going to state  $s'$  when performing action  $a$  in state  $s$ . An essential assumption

made in the MDP is that the dynamics of state evolution is *Markovian*, meaning that the distribution of the next states is conditionally independent of the past, given the current state.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  is a reward function.  $R_{s,s'}^a$  describes a *finite* payoff or reward obtained when the agent goes from state  $s$  to state  $s'$  as a result of executing action  $a$ . The reward can be either positive or negative, representing an utility or a cost, respectively.

The optimality objective is to find a way or a *policy* to maximize some measure of the long turn reward received. A (stationary) policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  is a mapping from states to action, which specifies an action to be taken for each state. The choice of action is independent of the time, depends only on the state. Given a policy, we can define a *value* function  $V_\pi(s)$  on the state space, which is the expected long run value an agent could expect to receive by choosing the action dedicated by the policy. A policy  $\pi_1$  is said to dominate another policy  $\pi_2$  if,  $V_{\pi_1}(s) \leq V_{\pi_2}(s)$  for any state  $s \in \mathcal{S}$ , and  $\exists s_1 \in \mathcal{S}$  such that  $V_{\pi_1}(s_1) < V_{\pi_2}(s_1)$ . A fundamental theorem [2] in MDP stated that there exists a stationary policy  $\pi^*$ , called the optimal policy, that dominates or has equal value to all other policies. The existence of such an optimal policy relies on the assumption that the expected long term reward, which is the objective function in the MDP, accumulates additively over time. That is to say, at each state, the optimal policy ranks the actions based on the sum of the expected rewards of the current time step and the optimal expected rewards of all subsequent steps.

To ensure the value function is well defined, one can limit the MDP to a finite number of time steps. In this case, the summation over rewards incurred in subsequent time steps terminates after a finite number of terms  $N$ , called the *horizon*, and the corresponding MDP is called a *finite horizon* MDP. The value of a policy  $\pi$ , starting from an initial state  $s_0$ , is

$$V_\pi^N(s) = \mathbb{E}[R(s_N) + \sum_{k=0}^{N-1} R(s_k, \pi(s_k), s_{k+1}) \mid s_0 = s] \quad (1)$$

where  $R(s_N)$  is a terminal reward for ending up with the final state  $s_N$ , and the expectation is taken with respect to the probability distribution of the Markov Chain  $\{s_0, s_1, \dots, s_N\}$  starting at the initial state  $s$ , with transition probability matrix  $P_{s_k, s_{k+1}}^{\pi(s_k)}$ . The optimal value function and the optimal policy is denoted by  $V^{*N}(s)$  and  $\pi^*(s)$ , respectively; that is,

$$V^{*N}(s) = \max_{\pi} v_{\pi}^N(s) \quad (2)$$

$$\pi^*(s) = \operatorname{argmax}_{\pi} v_{\pi}^N(s) \quad (3)$$

Despite the simple mathematical properties of the finite horizon MDPs, in many tasks, the reward is accumulated over an infinite (or indefinite) sequence of time steps. We refer this kind of tasks as the *infinite horizon* problems. There are three principal classes of infinite horizon problems.

- (a) **Discounted problems.** Here we introduce a discount factor  $\gamma$  with  $0 \leq \gamma < 1$ . The reward incurred at the  $t$ th transition is *discounted* by a factor  $\gamma^t$ . Then the value function over an infinite number of time steps is given by

$$V_{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, \pi(s_k), s_{k+1}) \mid s_0 = s\right] \quad (4)$$

In our assumption, the one step reward  $R_{ss'}^a$  is bounded from above by some constant, say,  $M$ . Therefore,  $v_{\pi}(s) \leq \sum_{t=0}^{\infty} \gamma^t M = \frac{M}{1-\gamma}$ , the infinite sum of decreasing geometric progression is finite for all policies  $\pi$  in all situations.

- (b) **Stochastic Shortest Path Problems.** Here  $\gamma = 1$  but we assume that there exists some additional termination state. Once the Markov chain reaches the termination state it remains there without any further rewards. The rewards (costs) associated with other states are negatively. In addition, the Markov chain is assumed to be such that termination is inevitable within finite number of steps, at least under an optimal policy. Thus, the problem is in effect a finite horizon one, but the length of horizon may be random. It can be shown that any discounted problems can be converted to a stochastic shortest path problem.

- (c) **Average reward problems.** Without the discount factor, the sum over an infinite sequence of rewards may be infinite, however, it turns out that in many problems the average reward per time step, given by

$$\tilde{V}_\pi^N \lim_{N \rightarrow \infty} \frac{1}{N} V_\pi^N(s) \quad (5)$$

where  $V_\pi^N(s)$  is the  $N$ -horizon value function of policy  $\pi$  starting at state  $s$ , is well defined as a limit and is finite.

The optimal value function  $V^*(s)$  can be shown to satisfy the well known *Bellman equation*

$$V^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}[R(s, a, s') + \gamma V^*(s')]. \quad (6)$$

## 1.2 Representations of MDPs

Exact solutions to MDP, such as value iteration [5], policy iteration [17], and linear programming [9], involve a *lookup table representation* of the value function, in the sense that the whole vector  $V(s)$  is kept in memory for each state  $s$ . The complexity of these algorithms are at least polynomial [29] in the size of the state space  $|\mathcal{S}|$  as well as the size of action space  $|\mathcal{A}|$ . However, the order of the polynomials is large enough that those exact algorithms are not efficient in practice. The computation requirements of large scale MDP are still overwhelming. In such problems a sub-optimal approximation solution using *compact representation* of MDPs needed to be used. compact representations for approximately solving MDPs. Widely used compact representations include

- Construct a low dimensional vector space representation of the value function by building a set of linear basis functions [3].
- Kernel (instance) based methods [28] that represent the value function as a convex combination of observed values in the simulation samples.
- Factored MDPs [6] construct a representation of the state space using a vector of state variables, and represent the transition models between state variables using a dynamic Bayesian network.
- Hierarchical representations [8, 11] of MDPs exploit the task structure, where the actions are temporally extended.
- Symbolic representation of MDPs express the state space as binary decision diagrams(BDD) and algebraic decision diagrams(ADD) [16].

However, finding a good compact representations for a given reinforcement learning (RL) task requires carefully hand-coding by a human designer, which can be quite difficult and time consuming. We further review recent developments in automatic discovery of efficient representations in MDPs. We elaborate the problems of automatically constructing structured kernel for kernel based RL, a popular approach to learning non-parametric approximations for value function. We provide algorithms for exploring a space of kernel structures which are built compositionally from base kernels using a context-free grammar, and greedy algorithms for searching over the structure space.

## 2 Solutions for a Lookup Table Representation

In this section, we review basic solutions to MDP with a lookup table representation of value function.

There are two fundamental classes of exact solution methods to MDPs. The first approach is based on iterative algorithms that use dynamic programming, whereas the second approach formulates an MDP as a linear program. These exact solutions require a perfect knowledge of the explicit models of the reward structure and transition probabilities of the system, which many not be available. Simulation methods based on Monte Carlo simulations, instead requires only sample transitions  $(s_t, a_t, r_t, s_{t+1})$  of the system.

The iterative algorithms typically employs the Bellman equation 6 to recursively relating the value of the current state to values of adjacent states. The form of Bellman equation motivates the introduction of two essential operators, also known as Bellman backup or dynamic programming backup operators in literature, that provide a convenient shorthand notation in expressions.

For any vector  $V = (V(1), \dots, V(|S|))$ , we consider the vector  $TV$  obtained by applying one iteration of right hand side of Bellman equation:

$$(TV)(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p_{ss'}^a (R(s, a, s') + \gamma V(s')) \quad (7)$$

and similarly, for any vector  $V$  and any stationary policy  $\pi$ , we consider the vector  $T_\pi V$  with components

$$(T_\pi V)(s) = \sum_{s' \in \mathcal{S}} p_{ss'}^{\pi(s)} (R(s, \pi(s), s') + \gamma V(s')) \quad (8)$$

Given a stationary policy  $\pi$ , we define the  $|\mathcal{S}| \times |\mathcal{S}|$  matrix  $P_\pi$  whose  $(i, j)$  entry is  $p_{i,j}^{\pi(i)}$ . Then we can re-write  $T_\pi V$  in matrix form as

$$T_\pi V = R_\pi + \gamma P_\pi V \quad (9)$$

where

$$R_\pi(s) = \sum_{s' \in \mathcal{S}} p_{ss'}^{\pi(s)} R(s, \pi(s), s') \quad (10)$$

We denote  $T^k$  and  $T_\pi^k$  as the operator obtained by applying the mapping  $T$  and  $T_\pi$  with themselves  $k$  times, respectively. It can be shown [3] that the following properties hold for  $T_\pi$  and  $T$ .

- (a) The optimal value vector  $V^*$  is the only solution to the equation  $V = T V$ .
- (b) We have  $\lim_{k \rightarrow \infty} T^k V = V^*$ . for every vector  $V$
- (c) A stationary policy is optimal if and only if  $T_\pi V^* = T V^*$ .
- (d) For every vector  $V$ , we have  $\lim_{k \rightarrow \infty} T_\pi^k V = V_\pi$ . And  $V_\pi$  is the only solution of the equation  $V = T_\pi V$
- (e) The operator  $T$  is a contraction mapping with respect to a weighted maximum norm. That is, there exists a vector  $\rho$  of size  $|\mathcal{S}|$  and a positive scalar  $\beta < 1$  such that

$$\|TV - TV'\|_\rho \leq \beta \|V - V'\|_\rho \quad (11)$$

for all vectors  $V$  and  $V'$ , and the weighted maximum norm is  $\|V\|_\rho = \max_{s \in \mathcal{S}} \frac{|V(s)|}{\rho(s)}$

### 2.1 Value Iteration

A principal method, called value iteration, for calculating the optimal value  $V^*$  is to generate a sequence  $T^k V$  starting from some vector  $V$  as  $\lim_{k \rightarrow \infty} T^k V = V^*$ . The value functions so computed are guaranteed to converge in the limit to the optimal value function. In the stochastic shortest path and average reward problems some additional assumptions for convergence are needed.

- *Finite ( $N$ ) horizon problem*: the algorithm always converge in  $N$  steps.
- *Infinite horizon problems with discount rewards*: the algorithm always converges to the unique optimal solution.
- *Stochastic shortest path problem*: the algorithm converges if there is a policy with positive probability of termination after at most finite time steps, regardless the initial state.
- *Average Reward problems*: the algorithm converges if every state can be reached from every other state in finite time step with positive probability for some policy.

---

**Algorithm 1** Value Iteration

---

```

1: Initial  $V_0$  arbitrarily for each state and  $t = 0$ 
2: repeat
3:   Compute  $V_t = TV_{t-1}$ 
4:   Compute Residual  $e_t = \|V_t - V_{t-1}\|_{max}$ 
5:    $t = t + 1$ 
6: until  $e_t < \epsilon$ 
7: return Greedy policy  $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}^a [R(s, a, s') + \gamma V_t(s')]$ 

```

---

A commonly used stopping rule is to set  $\epsilon = \epsilon' \frac{1-\gamma}{2\gamma}$ , which ensures the resulting value function is within  $\frac{\epsilon'}{2}$  of the optimal value function, and the resulting policy is  $\epsilon'$ -optimal [38].

The running time for each iteration in algorithm1 is  $O(|\mathcal{A}| |\mathcal{S}|^2)$ . The number of iterations until convergence it shown [22] to be polynomial in the size of the state space  $|\mathcal{S}|$  as well as the size of action space  $|\mathcal{A}|$ , which in turn makes value iteration polynomial in time. However, the order of the polynomials is nontrivial, thus in practice value iteration is usually inefficient.

## 2.2 Policy Iteration

Another widely used iterative algorithm is known as policy iteration [17]. At each iteration, the decision maker first carries out a *policy evaluation* phase, in which the value function associated with the current policy is computed, and a *policy improvement* phase, in which a greedy attempt is made to improve the current policy.

The basic policy iteration algorithm is described in algorithm 2, where policy evaluation step in-

---

**Algorithm 2** Policy Iteration

---

```

1: Let  $\pi_0$  be some random initial policy and  $t = 0$ 
2: repeat
3:   Policy Evaluation: compute  $V_{\pi_t}$  in equation 12.
4:   Policy Improvement:  $\pi_{t+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}^a (R_{ss'}^a + \gamma V_{\pi_t}(s'))$ , for all  $s \in \mathcal{S}$ 
5:    $t = t + 1$ 
6: until  $\pi_{t+1}(s) = \pi_t(s)$ , for all  $s \in \mathcal{S}$ 

```

---

volves solving a system of  $\mathcal{S}$  equations with  $\mathcal{S}$  unknowns. Let  $\rho$  be the invariant distribution of a Markov chain  $P_\pi$ , and let  $\mathcal{N}$  be the set of non-terminal states and  $\mathcal{T} = \mathcal{S} - \mathcal{N}$  be the set of zero reward termination states in stochastic shortest path problems.

$$\begin{aligned}
V_\pi(\mathcal{N}) &= (I - P_\pi(\mathcal{N}, \mathcal{N}))^{-1} (R_\pi(\mathcal{N}) + P_\pi(\mathcal{N}, \mathcal{T}) R_\pi(\mathcal{T})) && \text{Stochastic Shortest Path} \\
V_\pi &= (I - \gamma P_\pi)^{-1} R_\pi && \text{Discounted Reward} \\
\tilde{V}_\pi &= (1 - P_\pi)^{-1} (R_\pi - \rho) && \text{Average Reward}
\end{aligned} \tag{12}$$

For each iteration, policy evaluation phase can be performed in  $O(|\mathcal{S}|^3)$  arithmetic operations and policy improvement in  $O(|\mathcal{A}| |\mathcal{S}|^2)$  operations. When the number of states is large, it's usually preferable to carry out the policy evaluation phase by using iterative methods such as value iteration.

It can be shown that the policy iteration algorithm generates an improving sequence of policies and terminates with an optimal policy. There is no theoretical guarantees for the number of iterations required, yet policy iteration has been listed as one of the preferred solution method for MDP.

### 2.3 Linear Programming

A third approach to solve MDPs exactly is based on linear programming [9]. The primal linear program involves

$$\begin{aligned} \text{Variables:} & \quad V(s), \quad \forall s \in \mathcal{S} \\ \text{Minimize:} & \quad \sum \rho(s) V(s) \\ \text{Subject to:} & \quad V(s) \geq \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V_{\pi_t}(s')) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \end{aligned} \quad (13)$$

where  $\rho$  is known as the state relevance weight vector whose elements are all positive. There are  $|\mathcal{A}| |\mathcal{S}|$  constraints and  $|\mathcal{S}|$  variables, one constraint for each state  $s$  and action  $a$ . Thus, MDPs can be solve in polynomial time. A drawback of this algorithm is that it is typically slower than those iterative dynamic programming methods.

### 2.4 Temporal Difference Learning

In this subsection, we discuss an implementation of the Monte Carlo algorithm that incrementally updates the value function  $V(s)$  after each transition. We first express the value function as

$$\begin{aligned} V_{\pi}(s_t) &= \mathbb{E} \left[ \sum_{m=0}^{\infty} \gamma^m g(s_{t+m}, s_{t+m+1}) \right] \\ &= \mathbb{E} [g(s_t, s_{t+1}) + \gamma V_{\pi}(s_{t+1})] \end{aligned} \quad (14)$$

The Robbins-Monro stochastic approximation method for solving the above expectation equation takes the form

$$\begin{aligned} \hat{V}(s_t) &= (1 - \alpha_t) \hat{V}(s_t) + \alpha_t (g(s_t, s_{t+1}) + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \\ &= (1 - \alpha_t) \hat{V}(s_t) + \alpha_t d_t \end{aligned} \quad (15)$$

where  $\alpha_t \in (0, 1)$  is the learning rate and  $d_t = g(s_t, s_{t+1}) + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$  is called the temporal difference (TD) [36], representing the difference between an estimate  $g(s_t, \pi(s_t), s_{t+1}) + \gamma \hat{V}(s_{t+1})$  of the value function based on the one-step ahead simulated outcome of the current time step, and the current estimate  $\hat{V}(s_t)$ . Alternatively, we might fix a non-negative integer  $L$  and take into accounts the  $L + 1$ -step ahead simulated outcome,

$$V_{\pi}(s_t) = \mathbb{E} \left[ \sum_{m=0}^L \gamma^m g(s_{t+m}, s_{t+m+1}) + V_{\pi}(s_{t+L+1}) \right] \quad (16)$$

We cannot assume one  $L$  better than another in the absence of any special knowledge. For the sake of generality, we may combine a weighted average of  $L$ -step Bellman equation 16 over all possible  $L$ . We introduce a constant  $\lambda < 1$ , multiply Eq.16 by  $(1 - \lambda)\lambda^L$ , and sum over all non-negative  $L$ . We then have,

$$\begin{aligned} V_{\pi}(s_t) &= (1 - \lambda) \mathbb{E} \left[ \sum_{L=0}^{\infty} \lambda^L \left( \sum_{m=0}^L \gamma^m g(s_{t+m}, s_{t+m+1}) + V_{\pi}(s_{t+L+1}) \right) \right] \\ &= \mathbb{E} \left[ (1 - \lambda) \sum_{m=0}^{\infty} g(s_{t+m}, s_{t+m+1}) \sum_{L=m}^{\infty} \lambda^m + \sum_{L=0}^{\infty} (\lambda^L - \lambda^{L+1}) V_{\pi}(s_{t+L+1}) \right] \\ &= \mathbb{E} \left[ \sum_{m=0}^{\infty} \lambda^m \gamma^m d_{m+t} \right] + V_{\pi}(s_t) \end{aligned} \quad (17)$$

The resulting Robbins-Monro stochastic approximation method is then

$$\hat{V}(s_t) = (1 - \alpha_t) \hat{V}(s_t) + \alpha_t \sum_{m=t}^{\infty} (\lambda \gamma)^{m-t} d_m \quad (18)$$

The above equation provides a family of algorithms, one for each  $\lambda$ , and is known as TD( $\lambda$ ). The choice of  $\lambda$  reflects a trade-off between bias and variance in the Monte Carlo based approximation. The general conclusion from [35] shows that intermediate values of  $\lambda$  seem to work best in practise. Sutton [36] has shown that under TD(0), the temporal difference algorithm converges to the true value function  $V_\pi$ . Dayan [7] extended this result to the case of general  $\lambda$ .

A temporal difference based method for learning action values called Q-learning was introduced by Waktins [37]. Q-learning updates directly estimates of the Q-factors associated with an optimal policy, thereby avoiding the multiple policy evaluation phases of policy iteration. The following learning rule for learning the action value function  $Q(s, a)$  is used:

$$Q_{t+1}(s, a) = (1 - \alpha_t)Q_t(s, a) + \alpha_t(g(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_t(s', a')) \quad (19)$$

where  $s'$  and  $g(s, a, s')$  are generated from the pair  $(s, a)$  by simulation, according to the transition probability matrix  $P_{ss'}^a$ . Q-learning is sometimes referred to as an *off-policy* learning algorithm since it estimates the optimal action value function  $Q(s, a)$  while simulation the MDP using any policy. During simulation, a sequence of states is generated with the greedy actions provided by the current available Q-factors. It's possible that certain profitable actions are never explored. In practice, variants of Q-learning algorithms with parameters control the degree of exploration are introduced to ensure sufficient exploration during simulations.

### 3 Compact Representation of Markov Decision Processes

The solutions described in previous section require a lookup table representations of the value function  $V(s)$  with size  $|\mathcal{S}|$ . In environments with large discrete state space is large or even with continuous state spaces, the time complexity of the MDP solution algorithms makes them inefficient in practise. In this section, we review a variety of compact representations for approximately solving MDPs, including low dimensional vector space representations by constructing linear basis functions [3], instance based representations of value function using kernels in Hilbert space [28], factored representation [15], hierarchical representations [8, 11], and symbolic representations such as binary decision diagrams(BDD) and algebraic decision diagrams(ADD) [16]. All these approaches depend crucially on a choice of low dimensional compact representation of a MDP, and assume these are carefully provided by the human designer. The focus of this section is on approximation, rather than automatic representation discovery.

#### 3.1 Linear Value Function Approximation

In this subsection, we consider the policy evaluation phase for a single stationary policy  $\pi$ . Thus we suppress in our notation for the value functions the dependence on  $\pi$ . We approximate the value function  $V(s)$  with a linear architecture:

$$\hat{V}(s, w) = \phi(s)'w, \quad \forall s \in \mathcal{S} \quad (20)$$

where  $w$  is a weight vector and  $\phi(i)$  is an  $|\mathcal{D}|$ -dimensional feature vector associated with state  $s$ . That is, we represent the value function in a compact form  $V \approx \hat{V} = \Phi w$ , where  $\Phi$  is the  $|\mathcal{S}| \times |\mathcal{D}|$  matrix that has as rows the feature vectors  $\phi(s)$ ,  $s \in \mathcal{S}$ . Thus, we want to approximate the value function  $V$  with the subspace  $\mathcal{D}$  spanned by  $|\mathcal{D}|$  basis function, each of which is in the columns of  $\Phi$ . The rank of matrix  $\Phi$  is  $|\mathcal{D}|$ . Let  $\Pi$  be the projection operator on to the linear subspace, with respect to some norm  $\|\cdot\|_\rho$ :

$$\|V\|_\rho = \sqrt{\sum_{s \in \mathcal{S}} \rho_s V^2(s)}, \quad (21)$$

where  $\rho$  is a vector of positive components.  $\Pi V$  is the unique vector in the subspace that minimizes  $\|V - \Phi w\|_\rho$ .

$$\Pi V = \Phi w_\Phi \quad (22)$$

$$w_V = \underset{w \in \mathbb{R}^D}{\operatorname{argmin}} \|V - \Phi w\|_\rho^2 \quad (23)$$

By setting the gradient of Eq. 23 to 0, we have

$$\Pi = \Phi(\Phi' D_\rho \Phi)^{-1} D_\rho \quad (24)$$

where  $D_\rho$  is the  $|\mathcal{S}| \times |\mathcal{S}|$  diagonal matrix whose entries are  $\rho(s)$ . Now consider the Bellman backup operator  $T_\pi$  updating projected value functions,

$$\begin{aligned} \Phi w &= \Pi T_\pi(\Phi w) \\ \Phi w &= \Pi[R_\pi + \gamma P_\pi \Phi w] \end{aligned} \quad (25)$$

This equation is known as the projected Bellman's equation. And the solution  $\phi w_\Phi$  of this equation is the approximation to value function  $V_\pi$  in the subspace spanned by  $\Phi$ .  $w_\Phi$  satisfied

$$\begin{aligned} [\Phi' D_\rho (I - \gamma P_\pi) \Phi] w_\phi &= \Phi' D_\rho R_\pi \\ A w_\phi &= b \end{aligned} \quad (26)$$

and can be solved by matrix inversion  $w = A^{-1}b$  or other iterative algorithms. It can be shown that both mapping  $T_\pi$  and  $\Pi T_\pi$  are contraction [26] with respect to the weighted Euclidean norm  $\|\cdot\|_\rho$ , where  $\rho$  is the steady state probability vector of the Markov chain with transition probabilities  $P_\pi$ . Analog to value iteration, the so-called projected value iteration algorithm iteratively apply the contraction operator  $\Pi T_\pi$ , starting with some arbitrary vector  $w_0$

$$\Phi w_{t+1} = \Pi T_\pi(\Phi w_t) \quad (27)$$



However, the projected value iteration algorithm is not practical when  $|\mathcal{S}|$  is large since  $T_\pi(\Phi w_t)$  is of size  $|\mathcal{S}|$ , and the steady state probabilities  $\rho$  are assumed to be known.

Alternative way to solve equation 26 from simulation trajectories sampled from the Markov chain associated with policy  $\pi$ . After collecting  $t$  samples we have

$$\hat{A}_t = \frac{1}{t+1} \sum_{k=0}^t \phi(s_k)(\phi(s_k) - \gamma\phi(s_{k+1}))' \quad (28)$$

$$\hat{b}_t = \frac{1}{t+1} \sum_{k=0}^t \phi(s_k)R(s_k, s_{k+1}) \quad (29)$$

Given  $\hat{A}_t$  and  $\hat{b}_t$ , one can construct a simulation bases solution

$$w_t = \hat{A}_t^{-1} \hat{b}_t \quad (30)$$

This is known as the least square temporal difference (LSTD) method.

Similar to TD( $\lambda$ ) method, we can introduce a constant  $\lambda < 1$  and define

$$\hat{A}_t^\lambda = \frac{1}{t+1} \sum_{k=0}^t \phi(s_k) \sum_{m=k}^t \gamma^{m-k} \lambda^{m-k} (\phi(s_m) - \gamma\phi(s_{m+1}))' \quad (31)$$

$$\hat{b}_t^\lambda = \frac{1}{t+1} \sum_{k=0}^t \phi(s_k) \sum_{m=k}^t \gamma^{m-k} \lambda^{m-k} R(s_m, s_{m+1}) \quad (32)$$

the corresponding matrix inversion solution  $w_t = (\hat{A}_t^\lambda)^{-1} \hat{b}_t^\lambda$  is called the LSTD( $\lambda$ ) method.

### 3.2 Factored Markov Decision Processes

When some structure knowledge about the state space is known, one can construct a *factored MDP* representation of the state space using a vector of state variables, and represent the transition models between state variables using a dynamic Bayesian network. In this way, the value function can be approximated by a linear combination of basis functions, where each basis function involves only a small subset of the state variables. In particular, Guestrin et al [15] proposed an algorithm that generalize exact linear programming using basis functions  $\Phi$ .

$$\begin{aligned} \text{Variables: } & w_1, \dots, w_{|\mathcal{D}|} \\ \text{Minimize: } & \sum_s \rho(s) \sum_i w_i \phi_i s \\ \text{Subject to: } & \sum_i w_i \phi_i(s) \geq \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma \sum_i w_i \phi_i(s')) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \end{aligned} \quad (33)$$

where  $\rho$  is known as the state relevance weight vector whose elements are all positive. The number of variables in linear program has now been reduced from  $|\mathcal{S}|$  to  $|\mathcal{D}|$ , the number of basis function in sub-space  $\mathcal{D}$ . Without a factored representation of the state space, the number of constraints remains  $|\mathcal{S}| \times |\mathcal{A}|$ . For factored MDPs, the number of constraints can be reduced exponentially by exploiting conditional independence properties in the conditional probability table of the dynamic Bayesian network.

### 3.3 Kernel Based Reinforcement Learning

In the kernel based reinforcement learning (KBRL) algorithms [28, 18], value functions are approximated by a set of sample outcomes  $\{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{N_T}$ . Specifically, KBRL approximates the outcome of an action  $a$  from a given state  $s$  as the convex combination of sampled outcomes of that action, weighted by a function of the distance between  $s$  and sampled states. Then the Bellman backup operator is represented by an operator  $T_K$  on the samples:

$$\hat{V}(s) = T_K V(s) = \max_{a \in \mathcal{A}} \hat{Q}(s, a) \quad (34)$$

$$\hat{Q}(s, a) = \sum_{t \in \{t: a_t = a\}} K_a(s_t, s) [r_t + \gamma V(s_{t+1})] \quad (35)$$

where the summation is over a subset of indices  $t$  where  $a_t = a$ , and the kernel  $K_a(s_t, s)$  is normalized in the sense that for each state  $s$  and action  $a$ ,  $\sum_{t \in \{t: a_t = a\}} K_a(s_t, s) = 1$ .

Kernel-based reinforcement learning has several promising properties. First, the operator  $T_K$  has a unique fixed point. One can obtain an algorithm analog to value iteration to solve the MDP by iteratively applying  $T_K$ . Second, the fix point of this operator converges in probability to the true value function for the Gaussian Kernel:

$$K_a(s_t, s) = \exp\left[-\frac{d^2(s_t, s)}{2\sigma^2}\right] \quad (36)$$

when the number of samples  $N_T \rightarrow \infty$  and the bandwidth  $\sigma \rightarrow 0$ . The distance metric  $d(s_t, s)$  denotes the distance function. However, the time complexity of KBRL is  $N_T^2$ , which make it impractical when the sample size is large. To make it practical, Kveton [20] employs an unsupervised learning method to cluster the simulation samples onto  $k$  representative ones, and is able to compute the optimal policy in  $O(n)$  time assuming  $k \ll n$  a constant regardless  $n$ . Another advantage of the kernel based methods is the straightforward incorporation of the structure knowledge of the state space by using the structure kernel [21], where the kernel  $K_a(s_t, s)$  can be decomposed into a product of base kernels.

The kernel based algorithm defined above requires knowledge about the metric function of the state space. Alternatively, the Gaussian Process Temporal Difference (GPTD) [13] learning offers a Bayesian solution. Consider an episode in which a terminal state is reached at time step  $T + 1$ , with  $r_{T+1} = V(X_{T+1}) = 0$ . We have a generated model for the value function at state  $s_t$ :

$$V(s_t) = r_t + \gamma r_{t+1} + \dots + \gamma^{T-t} r_T - \epsilon_t \quad (37)$$

with  $\epsilon_t \sim \mathcal{N}(0, \sigma_t^2)$ . In a matrix form, we have

$$Z_T r_{1:T} = V_{1:T} + \epsilon_{1:T} \quad (38)$$

$$r_{1:T} = H_{T+1} V_{1:T} + \epsilon'_{1:T} \quad (39)$$

where

$$Z_T = \begin{bmatrix} 1 & \gamma & \gamma^2 & \dots & \gamma^T \\ 0 & 1 & \gamma & \dots & \gamma^{T-1} \\ \dots & & & \dots & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad H_T = Z_{T-1}^{-1} = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \dots & & & \dots & \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix} \quad (40)$$

Assuming a state-wise noise model with  $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ , we have  $\epsilon'_{1:T} \sim \mathcal{N}(0, \sigma^2 H_T H_T^T)$ .

Since both the value prior and the noise are Gaussian, so is the posterior distribution of the value conditioned on an observed sequence of rewards  $r_{1:T} = \{r_t\}_{t=1:T}$ . The joint distribution between a test point  $V(s^*)$  and the observed sequence is:

$$\begin{pmatrix} Z_T r_{1:T} \\ V(s^*) \end{pmatrix} = \mathcal{N} \left[ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} K_T & K_T(s^*) \\ K_T(s^*)^T & K(s^*, s^*) \end{bmatrix} \right] \quad (41)$$

where  $K_T$  denotes the  $T \times T$  matrix of the covariances evaluated at all pairs of observed states, and  $K_T(s^*)$  denotes the  $T \times 1$  vector of the covariances evaluated at pairs of observed state  $s_t$  and the test state  $s^*$ . The posterior mean and variance of the value at  $s^*$  are given, respectively, by

$$\hat{V}(s^*) = K_T(s^*)^T (K_T + \sigma I)^{-1} r_{1:T} \quad (42)$$

$$\text{VAR}(\hat{V}(s^*)) = K(s^*, s^*) - K_T(s^*)^T (K_T + \sigma I)^{-1} K_T(s^*) \quad (43)$$

### 3.4 Hierarchical Methods

Another approach to solving MDPs with large state spaces is to treat them as a hierarchical of task structures. In many cases, hierarchical solutions don't aim at providing an optimal value function to a MDP problem, but focus on gaining efficiency in execution time and learning time. Hierarchical learners are commonly structured as *delegation* behaviors. Feudal Q-learning [8] involves a hierarchy of learning problems, with higher level agents being masters and lower level agents being slaves. The highest level agent receives rewards  $r_t$  and states  $s_t$  from the external environment. It

learns a mapping from states  $s_t$  to some pre-defined intermediate commands and feeds the lower level slaves commands and corresponding rewards for taking actions that satisfy the command. The lower level agents learn a mapping from commands and states to external actions  $a_t$ . However, the set of intermediate commands and their associated reinforcement functions should be established in advance of the learning. Similarly, by assuming one can identify useful subgoals and define sub-tasks that achieve these subgoals, the MAXQ algorithms [11] that decompose the target MDP into a hierarchy of smaller MDPs were proposed. Using the MAXQ decomposition, the value function of the target MDP can be expressed as an additive combination of the value functions of the smaller MDPs. To amend restriction of human designed hierarchy, Mehta et al [25] further introduced an algorithm that can automatically discover the task hierarchy, given that the dynamic Bayesian networks associated with the action and reward models are provided, as well as successful sample trajectories following the optimal policy.

### 3.5 Symbolic Algorithms for Solving MDPs

We briefly discussed symbolic algorithms in this subsection. The key idea of symbolic algorithms is to compactly represent the MDP models (value function, transition probabilities, reward functions, etc) using decision diagrams, instead of using the table lookup representation. Similar to *aggregation* methods, these decision diagram representations cluster the states that share similar values. Instead of applying Bellman operator to each state, it is sufficient to update the subset of states with similar values as a whole at once, by just a single Bellman backup. This representation allows one to describe a value function as a function of the variables describing the domain and speeds up the value iteration based algorithms. However, these symbolic algorithms assume states in the MDP be factored. That is, the state space  $\mathcal{S}$  is factored into a set of  $d$  boolean state variables  $s = \{s_1, \dots, s_d\}$ . Although any finite valued non boolean variable can be split into a number of boolean variables, it often makes the new state space using decision diagram representation larger than the original one using the lookup table representation.

## 4 Representation Learning in Markov Decision Processes

In this section, we discuss methods for constructing compact representation of MDPs.

### 4.1 Feature Generation through Automatic Basis Construction

The policy evaluation phase can be viewed as solving systems of linear equation of the form  $Aw = b$ . The Krylov space method has long been among the most successful methods currently available for efficiently solving systems of linear equations. The  $k$ -order Krylov subspace is the linear subspace spanned by the image of  $b$  under the first  $k - 1$  powers of  $A$ , that is,

$$\text{Krylov}_k(A, b) = \text{span}\{b, Ab, A^2b, \dots, A^{k-1}b\} \quad (44)$$

For an MDP, typically we set  $b = R_\pi$ . The Krylov basis can be significantly accelerated by a computational trick called the Schultz expansion,

$$(1 - A)^{-1}b = (I + A + A^2 + \dots)b = \prod_{k=0}^{\infty} (I + A^{2^k})b \quad (45)$$

For example, we can compute the policy evaluation phase as follows:

$$V_\pi = (1 - \gamma P_\pi)^{-1}R_\pi = \prod_{k=0}^{\infty} (I + (\gamma P_\pi)^{2^k})R_\pi \quad (46)$$

Another way to construct basis automatically is based on the residual error in the current feature set [31]. Formally, if  $\Phi_k$  is the current set of basis functions, the Bellman error basis functions (BEBFs) add  $\phi_{k+1} = R + \gamma P\Phi_k w_{\Phi_k} - \Phi_k w_{\Phi_k}$  as the next basis function.

It's been shown [30] that a basis  $\Phi$  is not only useful in approximating value functions, but also induces a *low-dimensional* MDP. The induced approximate reward function  $R_\pi^\Phi$  and approximate transition function  $P_\pi^\Phi$  are defined as

$$R_\pi^\Phi = (\Phi' D_\rho \Phi)^{-1} \Phi' D_\rho R_\pi \quad (47)$$

$$P_\pi^\Phi = (\Phi' D_\rho \Phi)^{-1} \Phi' D_\rho P_\pi \Phi \quad (48)$$

where  $R_\pi^\Phi$  is the projection of the reward function  $R_\pi$  onto the column space of  $\Phi$ , with respect to  $\|\cdot\|_\rho$ . Similarly,  $P_\pi^\Phi$  is the least square solution to the system  $\Phi P_\pi^\Phi \approx P_\pi \Phi$ . The exact solution to this approximate MDP is the same as that given by the exact solution to the original MDP projected onto the basis  $\Phi$ .

Given basis constructed by Krylov space or BEBF methods with  $k$  basis functions, Mahadevan [23] propose the representation policy iteration algorithm, as described in Algorithm 3

---

#### Algorithm 3 Model-based representation policy iteration

---

- 1: Let  $\pi_0$  be arbitrary policy and  $t = 0$
- 2: **repeat**
- 3:   Construct basis matrix  $\Phi$
- 4:   From the MDP compute  $R_{\pi_t}^\Phi$  and  $P_{\pi_t}^\Phi$
- 5:   Find the solution to  $(1 - \gamma P_{\pi_t}^\Phi)w_\Phi = R_{\pi_t}^\Phi$
- 6:   Project solution back to the original state space  $V_{\pi_t}^\Phi = \Phi w_\Phi$ .
- 7:   Find the greedy policy  $\pi_{t+1}$  as in the policy improvement phase

$$\pi_{t+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V_{\pi_t}^\Phi(s')) \quad (49)$$

- 8:    $t = t + 1$
  - 9: **until**  $\pi_t = \pi_{t+1}$
  - 10: **return**  $\pi_{t+1}$
-

## 4.2 Feature Generation through Adaptive State Aggregation

Another basis construction algorithm [4] called the *adaptive state aggregation* partitions the original state space  $\mathcal{S}$  into a set of  $m$  subsets  $\mathcal{S}_1, \dots, \mathcal{S}_m$ , where  $\cup_{i=1}^m \mathcal{S}_i = \mathcal{S}$  and  $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ , for  $i \neq j$ . We can view state aggregation as a special form of basis matrix  $\Phi$ , where each column represents an indicator function for each cluster. At each iteration, the algorithm first carries out the regular value iteration to compute  $V^{k+1}$ , then corrects, rather than projects,  $V^{k+1}$  using the basis matrix

$$V^{k+1} = V^k + \Phi w_\Phi \quad (50)$$

where  $w_\Phi$  is the solution to the compact policy evaluation problem

$$w_\Phi = (I - \gamma P_\Pi^\Phi)^{-1} R_\Pi^\Phi \quad (51)$$

$$P_\Pi^\Phi = (\Phi' D_\rho \Phi)^{-1} \Phi' P_\pi \Phi \quad (52)$$

$$R_\Pi^\Phi = (\Phi' D_\rho \Phi)^{-1} \Phi' (T(V^k) - V^k) \quad (53)$$

To create the basis  $\Phi$  automatically, Keller [19] proposed to use neighborhood component analysis (NCA), a supervised learning algorithm with the state  $s$  as the input attributes, and the Bellman error or the temporal difference error as the supervised signal. In this way, NCA places basis function in the lower-dimensional space. The new lower dimensional features are then added as new features for the linear function approximator.

## 4.3 Structure Learning in Factored MDPs

---

### Algorithm 4 Structure Learning Algorithm for factored MDP

---

- 1: Initialization
  - 2: **for** each time step  $t$  **do**
  - 3:   Given  $s, \pi_{t-1}(s)$ , observe  $s'$  and  $r$
  - 4:   Update the factored representation of reward  $\text{Fact}(R_t)$  and transition  $\text{Fact}(P_t)$  functions.
  - 5:   Learn a policy  $\pi_t$  using structure value iteration or algorithms for factored MDP.
  - 6: **end for**
- 

Factored MDPs [6, 15] compactly represent the transition and reward functions of a MDP using dynamic Bayesian networks (DBNs). Efficient algorithms based linear program were developed even when the state space is large. However, they require a complete knowledge of the transition and reward functions of the problem in advance. Structure learning algorithms [10], as sketched in Algorithm 4 has been proposed to learn these functions by simulation trials, where decision tree induction algorithms are used to learn a factor representation of the reward and transition functions. Given the sample transitions  $\{s_t, a_t, r_t, s_{t+1}\}$  observed in a MDP system, decision tree induction algorithms learn the compact reward model with  $\{s_t\}$  being example attributes and  $\{r_t\}$  being example labels, and learn a conditional probabilities table representation of the transition model with  $\{s_t\}$  being example attributes and  $\{s_{t+1}\}$  being example labels. A  $\chi^2$  test is used to detect the independence between two random variables. After a factored representation of the model is learned incrementally, the improved policy can be obtained by an incremental version of structured value iteration [6]. At the next iteration, the agent will follow the  $\epsilon$ -greedy variant of the updated policy and generate new simulation samples. The algorithm will again update its factored representation for the model.

## 4.4 Structure Discovery through Compositional Kernel Search

Unlike the parametric linear function approximation using basis  $\Phi$ , Kernel-based reinforcement learning (KBRL) [28, 33] is a popular approach to learning a non-parametric representation of the value function, where the similarities between two states are captured by a kernel  $K_a(s, s')$ . In problems where the state space is factored and  $s$  can be expressed as a set of state variables, among which there exists some conditional independencies, structured kernels [21] should be used to capture the independent relationships. When the conditional independencies between the state variables are unknown in advance, kernel learning techniques need to be employed. By defining a space of kernel structures which are built compositionally from a context free grammar, we proposed

a greedy search algorithm based on the previous works [14, 12] to search over the grammar and automatically choose the decomposition structure from raw data by evaluation only a small fraction of all structures. We plan to demonstrate how the learned structure can represent and approximate the original RL problem in terms of compactness and efficiency, and evaluate our method on a synthetic problem and compare it to other RL baselines.

## 5 Related Work and Future Challenges

The representation learning methods described in this report can be applied to build representations from sampled examples over a large variety of problems in AI. They are also close related to recent work on manifold learning [34, 1] and spectral learning [27], which have largely been applied to nonlinear dimensionality reduction and semi-supervised learning problems on graphs. However, learning the compact MDP representation introduces new challenges not represented in supervised learning and dimensionality reduction, as the set of training examples is not available as a batch, but must be collected through active exploration of the state space. Another challenge for representation learning in reinforcement learning is how well a compact representation transfers from one problem to another.

## References

- [1] BELKIN, M., AND NIYOGI, P. Semi-supervised learning on riemannian manifolds. *Machine Learning* 56, 1-3 (2004), 209–239.
- [2] BELLMAN, R. E. *Dynamic Programming*. Princeton University Press, 1957.
- [3] BERTSEKAS, D. P. *Dynamic Programming and Optimal Control*, 2nd ed. Athena Scientific, 2000.
- [4] BERTSEKAS, D. P., AND CASTAÑON, D. A. Adaptive Aggregation Methods for Infinite Horizon Dynamic Programming. *IEEE Trans. on Automatic Control* 34, 6 (1989), 589–598.
- [5] BERTSEKAS, D. P., AND TSITSIKLIS, J. N. *Neuro-Dynamic Programming*, 1st ed. Athena Scientific, 1996.
- [6] BOUTILIER, C., DEARDEN, R., AND GOLDSZMIDT, M. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121 (1999), 2000.
- [7] DAYAN, P. The convergence of td( $\lambda$ ) for general  $\lambda$ . *Machine Learning* 8 (1992), 341–362.
- [8] DAYAN, P., AND HINTON, G. E. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems* 5 (1993), Morgan Kaufmann, pp. 271–278.
- [9] DE FARIAS, D. P., AND VAN ROY, B. The linear programming approach to approximate dynamic programming. *Oper. Res.* 51, 6 (Nov. 2003), 850–865.
- [10] DEGRIS, T., SIGAUD, O., AND WUILLEMIN, P.-H. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning* (2006), W. W. Cohen and A. Moore, Eds., vol. 148 of *ACM International Conference Proceeding Series*, ACM, pp. 257–264.
- [11] DIETTERICH, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13 (2000), 227–303.
- [12] DUVENAUD, D. K., LLOYD, J. R., GROSSE, R., TENENBAUM, J. B., AND GHAHRAMANI, Z. Structure discovery in nonparametric regression through compositional kernel search. *CoRR* (2013).
- [13] ENGEL, Y., MANNOR, S., AND MEIR, R. Reinforcement learning with gaussian processes. In *Proceedings of the 22Nd International Conference on Machine Learning* (New York, NY, USA, 2005), ICML '05, ACM, pp. 201–208.
- [14] GROSSE, R. B., SALAKHUTDINOV, R., FREEMAN, W. T., AND TENENBAUM, J. B. Exploiting compositionality to explore a large space of model structures. Tech. rep., MIT, 2012.
- [15] GUESTRIN, C., KOLLER, D., PARR, R., AND VENKATARAMAN, S. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research (JAIR)* 19 (2003), 399–468.
- [16] HOEY, J., ST-AUBIN, R., HU, A., AND BOUTILIER, C. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (1999), Morgan Kaufmann, pp. 279–288.
- [17] HOWARD, R. A. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

- [18] JONG, N., AND STONE, P. Kernel-based models for reinforcement learning in continuous state spaces. In *ICML workshop on Kernel Machines and Reinforcement Learning* (June 2006).
- [19] KELLER, P. W., MANNOR, S., AND PRECUP, D. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning* (New York, NY, USA, 2006), ACM, pp. 449–456.
- [20] KVETON, B., AND THEOCHAROUS, G. Kernel-based reinforcement learning on representative states. In *Association for the Advancement of Artificial Intelligence* (2012).
- [21] KVETON, B., AND THEOCHAROUS, G. Structured kernel-based reinforcement learning. In *Association for the Advancement of Artificial Intelligence* (2013).
- [22] LITTMAN, M. L., DEAN, T. L., AND KAEHLING, L. P. On the complexity of solving markov decision problems. In *IN PROC. OF THE ELEVENTH INTERNATIONAL CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE* (1995), pp. 394–402.
- [23] MAHADEVAN, S. Representation policy iteration. *CoRR abs/1207.1408* (2012).
- [24] MAUSAM, AND KOLOBOV, A. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.
- [25] MEHTA, N., RAY, S., TADEPALLI, P., AND DIETTERICH, T. G. Automatic discovery and transfer of task hierarchies in reinforcement learning. *AI Magazine* 32, 1 (2011), 35–50.
- [26] MUNOS, R. Error bounds for approximate policy iteration. In *International Conference on Machine Learning* (2003).
- [27] NARAYANAN, H., BELKIN, M., AND NIYOGI, P. On the relation between low density separation, spectral clustering and graph cuts. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, Cambridge, MA, 2007.
- [28] ORMONEIT, D., AND SEN, S. Kernel-based reinforcement learning. In *Machine Learning* (1999), pp. 161–178.
- [29] PAPADIMITRIOU, C., AND TSITSIKLIS, J. N. The complexity of markov decision processes. *Math. Oper. Res.* 12, 3 (Aug. 1987), 441–450.
- [30] PARR, R., LI, L., TAYLOR, G., PAINTER-WAKEFIELD, C., AND LITTMAN, M. L. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning* (New York, NY, USA, 2008), ACM, pp. 752–759.
- [31] PARR, R., PAINTER-WAKEFIELD, C., LI, L., AND LITTMAN, M. L. Analyzing feature generation for value-function approximation. In *ICML* (2007), pp. 737–744.
- [32] PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 1994.
- [33] RASMUSSEN, C. E., AND KUSS, M. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16* (2004), MIT Press, pp. 751–759.
- [34] ROWEIS, S. T., AND SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE* 290 (2000), 2323–2326.
- [35] SINGH, S. P., AND DAYAN, P. Analytical mean squared error curves in temporal difference learning. In *NIPS* (1996), MIT Press, pp. 1054–1060.
- [36] SUTTON, R. S. Learning to predict by the methods of temporal differences. In *MACHINE LEARNING* (1988), Kluwer Academic Publishers, pp. 9–44.
- [37] WATKINS, C. J. C. H., AND DAYAN, P. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.
- [38] WILLIAMS, R., AND BAIRD, L. C. Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep., Northester University, College of Computer Science, 1993.